/\*

A peak element is an element that is greater than its neighbors.

Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that num[-1] = num[n] = -∞.

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

way-1:一次遍历吧

way-2:直接找最大

\*\*\*\*\*\*\*\*\*\*\*\*\*二分查找原理：

根据给出的条件： num[i] != num[i+1], 相邻两个元素不相等。运用二分查找原理。

（1）如果 num[i-1] < num[i] > num[i+1], 则num[i] 就是 peak

（2）如果 num[i-1] < num[i] < num[i+1], 则 num[i+1...n-1] 必定包含一个 peak

（3）如果 num[i-1] > num[i] > num[i+1], 则num[0...i-1] 必定包含一个 peak

（4）如果 num[i-1] > num[i] < num[i+1], 则 两边都有一个 peak

继续优化一下，通过上面仔细观察一下：

（1）如果 num[i-1] < num[i] > num[i+1], 则num[i] 就是 peak

（2）如果 num[i-1] < num[i] , 则 num[i+1...n-1] 必定包含一个 peak，left指向mid+1

（3）如果 num[i-1] > num[i] , 则num[0...i-1] 必定包含一个 peak,right指向mid-1

way-3:二分查找找最大（递归）

way-4:二分查找找最大（迭代）

\*/

class Solution {

public:

int helper(vector<int>& nums,int l,int r)

{

int m=(l+r)/2;

if(l==r)

return l;

if(nums[m]>nums[m+1])

return helper(nums,l,m);

else

return helper(nums,m+1,r);

}

int findPeakElement(vector<int>& nums)

{

//way-1

/\*

for(int i=0;i<nums.size()-1;i++)

{

if(i==0)

{

if(nums.size()==1 || nums[i]>nums[i+1])

return 0;

}

else

{

if(nums[i]>nums[i-1] && nums[i]>nums[i+1])

return i;

}

}

return nums.size()-1;

\*/

//way-2

/\*

vector<int>::iterator it=max\_element(nums.begin(),nums.end());

return it-nums.begin();

\*/

//way-3

/\*

return helper(nums,0,nums.size()-1);

\*/

//way-4

int l=0;

int r=nums.size()-1;

while(1)

{

int m=(l+r)/2;

if(l==r)

return l;

if(nums[m]>nums[m+1])

r=m;

else

l=m+1;

}

return l;

}

};